Санкт-Петербургский государственный университет
Факультет прикладной математики - процессов управления
Кафедра математической теории игр и статистических решений

Гумеров Тагир Арсланович

Выпускная квалификационная работа бакалавра

Игровые модели формирования сетевого взаимодействия

Направление 010400.62

Прикладная математика, фундаментальная информатика и основы программирования

Научный руководитель,
доктор физ.-мат. наук,
профессор
Петросян Л. А.

Санкт-Петербург

2017

Содержание

Введение 3

Глава 1. Сетевые игры 5

Глава 2. Модель Лагераса и Зайна 7

2.1. Особенности игры 7

2.2. Ограничения на функцию выигрыша 8

2.3. Равновесие по Нэшу в чистых стратегиях 9

2.4. Пороговый граф 12

2.5. Многошаговая игра 15

Глава 3. Программная реализация модели Лагераса и Зайна 22

3.1. Формат представления данных 22

3.2. Реализованные функции 23

3.3. Примеры 27

Глава 4. Кооперативный вариант игры 32

4.1. Теоретико-игровая модель кооперативной игры 32

4.2. Проверка непустоты c-ядра 34

Список литературы 38

Приложение 39

Введение

Данная работа посвящена анализу моделей сетевых игр в динамике и процессу их формирования.

Andreas Lagerås, David Seim [1] предлагают модель сетевой игры с одновременным и независимым выбором стратегий и этапом формирования сети.

Авторы накладывают условия на функцию выигрыша, обеспечивающие существование и единственность равновесия по Нэшу в чистых стратегиях для сетей общего вида. Далее рассматривается класс графов, называемых пороговыми графами. Общее множество сетей сужается до этого класса, что позволяет сделать некоторые качественные утверждения о ситуации равновесия. Для строгого описания и последующего применения сведений из теории графов были изучены материалы из учебных пособий [4], [5]. Примечательно то, что свойства пороговых графов согласуются с литературой по социологии. В заключительной части статьи рассматривается многошаговая игра, в ходе которой помимо выбора игроками стратегий, влияющих на функцию выигрыша, случайно выбранный игрок должен изменить структуру сети. Доказывается, что, во-первых, при эндогенном формировании сети класс пороговых графов является поглощаемым, т.е. сеть не сможет покинуть этот класс. Во-вторых, любая сеть, при вероятности формирования связи большей, чем некоторая наперед заданная величина ε, сходится к пороговому графу с вероятностью единица.

В ходе данной работы был реализован программный алгоритм формирования сети по свойствам функции выигрыша, при которых игроки под действием внешнего процесса принимают решения об оптимальном

создании или удалении связей.

Для некооперативного варианта игры доказана непустота c-ядра для игр на полных сетях. К сожалению, построить как само c-ядро, так и получить значения характеристической функции не представляется возможным из-за общего вида ограничений, накладываемых на функции выигрыша.

Глава 1. Сетевые игры

В этой главе будут описаны основные сведения из теории игр, также введена сетевая игра в классическом виде.

Рассмотрим игру в нормальной форме Γ:

$$\Gamma = (N, \{X_i\}_{i \in N}, \{u_i\}_{i \in N}), \tag{1}$$

где $N = \{1, \ldots, n\}$ — конечное множество игроков, X_i — множество стратегий игрока $i \in N$, u_i — выигрыш игрока i.

Ситуацией в игре назовем вектор из стратегий игроков:

$$x = (x_1, \ldots, x_n) \in X \tag{2}$$

$\forall i \in N \colon x_i \in X_i$, где X — пространство ситуаций игры Γ, заданное как декартово произведение множеств стратегий игроков $X = X_1 \times \ldots \times X_n$.

Понятие сетевой игры подразумевает под собой наличие объекта, задающего правило взаимодействия игроков. (N, g) будем называть сетью. В ней N — множество вершин сети, совпадающее с множеством игроков. $g = N \times N$ — матрица сопряженности сети, а $g \in G$ — множество всевозможных сетей. Каждой паре $(i, j) \in N \times N$ взаимно-однозначно соответствует элемент матрицы g_{ij}. Будут рассматриваться сети, в которых образование связи между игроками — симметричная операция, игрок не может быть связан в сети сам с собой.

Таким образом, $g = \{g_{ij}\}_{i,j \in N}$ — симметричная матрица, состоящая из элементов g_{ij}, которые интерпретируются как значимость связи между игроками i и j, причем $g_{ii} = 0, \ \forall i \in N$. Игроки i и j не связаны между собой тогда и только тогда, когда $g_{ij} = 0$. Игроков $i, \ j \colon g_{ij} \neq 0$ будем называть соседями.

Введем несколько обозначений. $N(i) = \{j \in N : g_{ij} \neq 0\}$ — окрестность игрока i, то есть множество его соседей. $d(i) = |N(i)|$ будем называть степенью игрока. g_i — вектор связей игрока i, т.е. строка матрицы g, а G_i — множество всевозможных связей игрока i: $g_i \in G_i$.

Выигрыш игрока i зависит от ситуации в игре и текущей структуры сети. Причем, на его выигрыш не будет влиять наличие связи между игроками j и k, но будет влиять его собственная связь с игроками j и k.

Таким образом, функция выигрыша определена на пространстве ситуаций игры Γ, множестве всевозможных связей игрока i в сети и принимает вещественное значение:

$$u_i : X \times G_i \to \mathbb{R}. \tag{3}$$

На самом деле в сетевой игре на выигрыш игрока влияет вся сеть посредством выбора игроками их стратегий, но это происходит неявно.

Глава 2. Модель Лагераса и Зайна

2.1. Особенности игры

Расширим игру (1) до рассматриваемой модели.

Множество игроков $N = \{1, \ldots, n\}$ — конечно, $\forall i, j \in N$:

1. $X_i = \mathbb{R}_+$ — множество стратегий игрока i является множеством неотрицательных вещественных чисел. Будем отождествлять каждую конкретную стратегию с некоторым «усилием» игрока, которое он прикладывает, для получения выигрыша.

2. Пространство всех ситуаций в игре $X = \mathbb{R}_+^n$.

3. $g_{ij} = \{0, 1\}$, т.е. сеть невзвешенная. Это означает, что два игрока либо связаны между собой, либо нет, без градаций «силы» связи.

4. Функцию выигрыша можно задать в виде функции однородной полезности $u^o \colon \mathbb{R}_+^{n+1} \times G \to \mathbb{R}$:

$$u_i(x_1, \ldots, x_n; g_i) = u^o(x_i, g_{i1}x_1, \ldots, g_{in}x_n), \tag{4}$$

причем функция u^o симметрична по всем аргументам кроме первого. Это означает, что u^o будет принимать одно и то же значение, независимо от порядка учета остальных игроков.

Эквивалентные множества усилий и симметричность однородной функции полезности означают, что игра симметрична. Это замечание важно, т.к. позволяет рассматривать симметричные равновесия по Нэшу. То есть те равновесия, в которых игроки используют одинаковые стратегии.

Здесь важно обратить внимание на то, что отсутствие связи между игроками i и j оказывает такой же эффект на функцию u_i, как и в случае, если бы связь существовала, но $x_j = 0$.

2.2. Ограничения на функцию выигрыша

В этом разделе рассмотрим полный граф, матрицу смежности которого обозначим за K, K_i — i-ая строка полного графа. Функция выигрыша u_i должна быть трижды непрерывно дифференцируема.

Пусть $A(x)$ — функциональная матрица, определенная на некотором множестве $D \subseteq \mathbb{R}^n$.

Определение 1. *Матрица $A(x)$ строго положительно определена, если существует $\varepsilon > 0$ такое, что $y^T A(x) y \geq \varepsilon ||y||^2$ выполняется для всех $y \in \mathbb{R}^n \setminus \{0\}$ и для всех $x \in D$.*

Определение 2. *Матрица называется Z-матрицей, если все ее элементы, не лежащие на диагонали, не положительны.*

Введем обозначения:

предельная полезность в собственных стратегиях:

$$h_i(x; K_i) = \frac{\partial}{\partial x_i} u_i(x; K_i); \tag{5}$$

вектор из предельных усилий всех игроков, взятый с минусом:

$$\overline{h}(x; K) = -(h_1(x; K_1), \ldots, h_n(x; K_n)); \tag{6}$$

якобиан $\overline{h}$:

$$J(x; K) = - \begin{pmatrix} \frac{\partial}{\partial x_1} h_1(x; K_1) & \cdots & \frac{\partial}{\partial x_n} h_1(x; K_1) \\ \vdots & \ddots & \vdots \\ \frac{\partial}{\partial x_1} h_n(x; K_n) & \cdots & \frac{\partial}{\partial x_n} h_n(x; K_n) \end{pmatrix}. \tag{7}$$

u_i должна удовлетворять следующим условиям:

1. Считаем, что всегда выгодно прикладывать усилия:

$$h_i(x_1, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_n; K_i) > 0 \qquad \forall i \in N.$$

2. Предельные усилия игрока i не уменьшаются от действий соседа:

$$\frac{\partial}{\partial x_j} h_i(x; K_i) \geq 0 \qquad \forall i \neq j \in N,$$

причем равенство будет только при $x_j = 0$. Это значит, что при увеличении усилий соседей, функция выигрыша i-ого игрока будет только увеличиваться.

3. Ограничим потенциальные эффекты обратных связей игроков:

$$J(x) \text{ — положительно определен.}$$

Это предположение сделано для того, чтобы из-за условия (2), игроки не устремляли свои усилия к бесконечности. В частности будем полагать, что предельная полезность в собственных усилиях уменьшается: $\frac{\partial}{\partial x_i} h_i(x; K_i) < 0 \ \forall i \in N$. Также это условие будет гарантировать, что сколько бы усилий не приложили игроки, функция будет оставаться вогнутой.

4. Добавим техническое условие, которое позволит расширить полный граф до произвольного графа:

$$\frac{\partial^2}{\partial x_k \partial x_j} h_i(x; K_i) \geq 0 \qquad \forall i \neq k, \forall j.$$

2.3. Равновесие по Нэшу в чистых стратегиях

Лемма 1. *Если условия (1-4) из раздела (2.2) выполнены, то они выполняются при замене K на произвольную матрицу смежности g.*

Доказательство. Пусть условия (1-4) выполнены. Тогда для любого игрока i: $g_{ij} = 0$ эквивалентно $x_j = 0$, на что было обращено внимание в конце раздела (2.1). Условия выполняются для $x \in \mathbb{R}^n_+$, значит они выполняются и когда некоторые из элементов $x_i = 0$. Условия (1,2,4) верны для графов всегда. Значит, требуется заменить $\mathbb{R}^n \setminus \{0\}$ на $\mathbb{R}^n_+ \setminus \{0\}$ в определении строгой положительной определенности при некоторых ограничениях на $A(x)$. Введем лемму, в которой вводятся ограничения на $A(x)$, после чего завершим доказательство.

Лемма 2. *Пусть $A(x)$ — Z-матрица, тогда $A(x)$ — строго положительно определенна тогда и только тогда, когда существует $\varepsilon > 0$ такое, что $z^T A(x) z \geq \varepsilon ||z||^2$ выполняется для всех $z \in \mathbb{R}^n_+ \setminus \{0\}$ и для всех $x \in D$.*

Доказательство. Необходимость: пусть матрица $A(x)$ — положительно определенна, тогда по определению условие выполняется для $z \in \mathbb{R}^n$, и т.к. $\mathbb{R}^n_+ \subseteq \mathbb{R}^n$, оно будет выполняется и для $z \in \mathbb{R}^n_+$.

Достаточность: разложим произвольный вектор $y = (y_1, \ldots, y_n)^T$ на два вектора: $z = (z_1, \ldots, z_n)^T$ и $s = (s_1, \ldots, s_n)^T$: $s_i = sign(y_i),\ z_i = |y_i|,\ \forall i \in \{1, \ldots, n\},\ \forall x \in D,\ A(x) = \{a_{ij}\}_{i,j \in \{1,\ldots,n\}}$. Докажем утверждение в общем виде.

Матрица является Z-матрицей: $a_{ij} \leq 0\ \forall i \neq j$, по условию леммы:

$$y^T A(x) y - \varepsilon ||y||^2 =$$

$$= \sum_i a_{ii} y_i^2 + \sum_{i \neq j} a_{ij} y_i y_j - \varepsilon ||y||^2 =$$

где $\sum_{i \neq j} a_{ij} y_i y_j = \sum_{i \neq j} a_{ij} s_i s_j z_i z_j = \sum_{i \neq j, s_i = s_j} a_{ij} z_i z_j - \sum_{i \neq j, s_i \neq s_j} a_{ij} z_i z_j$

$$= \sum_i a_{ii} z_i^2 + \sum_{i \neq j, s_i = s_j} a_{ij} z_i z_j - \sum_{i \neq j, s_i \neq s_j} a_{ij} z_i z_j - \varepsilon ||z||^2 \geq$$

$$\geq \sum_i a_{ii} z_i^2 + \sum_{i \neq j, s_i = s_j} a_{ij} z_i z_j + \sum_{i \neq j, s_i \neq s_j} a_{ij} z_i z_j - \varepsilon ||z||^2 =$$
$$= \sum_i a_{ii} z_i^2 + \sum_{i \neq j} a_{ij} z_i z_j - \varepsilon ||z||^2 =$$
$$= z^T A(x) z - \varepsilon ||z||^2$$

Таким образом, $v^T A(x) v \geq \varepsilon ||v||^2, \ \forall v \in \mathbb{R}^n_+ \setminus \{0\} \Rightarrow$ $v^T A(x) v \geq \varepsilon ||v||^2, \ \forall v \in \mathbb{R}^n \setminus \{0\}$ и $A(x)$ — строго положительно определенна. □

Продолжение доказательства Леммы 1.

Для фиксированного x:

$a_{ij} = -\dfrac{\partial}{\partial x_j} h_i(x; K_i)$, $b_{ij} = -\dfrac{\partial}{\partial x_j} h_i(x; g_i)$, причем, как было показано, a_{ij} удовлетворяет условиям Леммы 2. Требуется показать, что и b_{ij} также удовлетворяет этим условиям.

По условию (4), $b_{ij} \leq a_{ij}$. Это верно из-за того, что отсутствие соседа, т.е. связи, в сети с матрицей смежности g эквивалентно тому, что этот сосед не прилагает усилия.

$\sum_{i,j} b_{ij} z_i z_j > \sum_{i,j} a_{ij} z_i z_j, \forall z \in \mathbb{R}^n_+$, откуда по Лемме 2 следует, что условие (3) выполнено для всех графов. □

Теорема 1. *Существует единственное равновесие по Нэшу в чистых стратегиях для некооперативной игры на любой сети, заданной матрицей смежности g с функцией выигрыша, удовлетворяющей условиям (1-4).*

Доказательство. Из условия (3), по Теореме 3.1 из [2] $\overline{h}$ строго монотонна. А из Теоремы 5.1 из [3] следует существование и единственность равновесие по Нэшу в чистых стратегиях. □

2.4. Пороговый граф

Рассмотрим класс графов, называемый пороговыми графами. Эти графы часто называют графами иерархии, из-за их ключевого свойства — иерархии вершин по их степени.

Это свойство хорошо иллюстрирует итерационный способ их построения. Граф, содержащий одну вершину — пороговый. Если к пороговому графу добавить изолированную вершину, он останется в классе пороговых графов. Если к пороговому графу добавить вершину и построить все инцидентные этой вершине ребра, граф также останется в классе пороговых графов. Вершина, соединенная со всеми остальными вершинами ребром, называется доминирующей. Из описания ясно, что один из способом задания порогового графа — последовательность из 2 различный символов длины n. Один символ отвечает за добавление изолированной вершины, второй - за добавление доминирующей вершины, пусть это будет 0 и 1 соответственно. Задание первой вершины графа можно совершить любым символом (в литературе [4] первый символ игнорируют, мы же будем обозначать его нулем).

На рис. 1 приведен пороговый граф, полученный итерационным алгоритмом добавления новой вершины к пороговому графу. Последовательность из нулей и единиц для примера: $(0, 1, 1, 1, 0, 0, 1, 0, 1, 0)$.

Ведущая вершина - вершина, имеющая наибольшее количество связей, в примере это вершина 8. Вершина 8 связана со всеми вершинами, имеющими хотя бы одну связь.

Для любого графа можно ввести разбиение по степени следующим образом. Пусть $N = \{1, \ldots, n\}$ — множество всех вершин графа, D — количество градаций степеней вершин в графе.

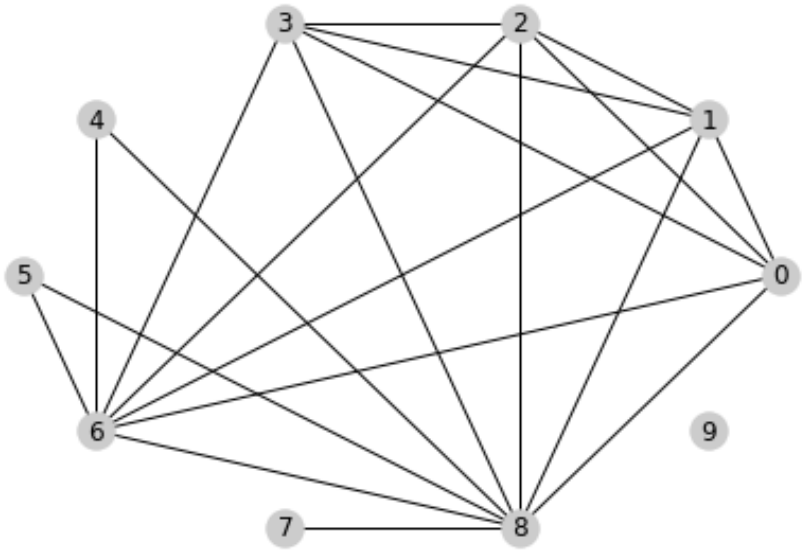


Рис. 1: Пороговый граф.

Зададим разбиение всех вершин на непересекающиеся множества:

$$N = \left\{ \bigcup_{j=1}^{D} Z_j \middle| \forall i \in Z_j : d(i) = d_j;\ Z_k \cap Z_l = \varnothing,\ \forall k \neq l \right\}, \tag{8}$$

где $d_1 > d_2 > \ldots > d_D$ — степени вершин в соответствующем блоке.

Вернемся к примеру. Стоит обратить внимание, что в нем нумерация вершин идет с 0, $N = \{0, \ldots, n-1\}$. Степени вершин данного графа (5, 5, 5, 5, 2, 2, 7, 1, 8, 0). Сгруппируем вершины по степеням и отсортируем их в обратном порядке по значению степени: $\{8, 6, (0, 1, 2, 3), (4, 5), 7, 9\}$, соответственно степени внутри каждого блока образуют убывающую последовательность $\{8, 7, 5, 2, 1, 0\}$.

Очевидно, что пустой и полный графы - пороговые. Действительно, если все вершины будут добавляться как изолированные, получим пустой граф. Иначе, если все вершины при добавлении будут соединены со всеми вершинами графа, получим полный граф.

За D^+ примем номер блока, имеющего минимальную ненулевую степень вершин, т.е. $D^+ = D$, если в графе нет изолированных вершин, и

$D^+ = D - 1$ в ином случае.

Определение 3. *Не пустой и не полный граф - пороговый граф тогда и только тогда, когда* $\forall i \in Z_j$ *соединен с* $\forall k \in \{Z_1, \ldots, Z_{D^+ + 1 - j}\}$.

Из определения следует, что в пороговом графе расстояние между двумя вершинами либо равно 1, либо 2, либо эти две вершины не связаны.

Теорема 2. *На пороговом графе ранжирование усилий в симметричном равновесии Нэша равно ранжированию степеней.*

Доказательство. Рассмотрим 2 вершины i и j, они одновременно являются и игроками. Если $d(i) = d(j)$, то обе вершины принадлежат одному блоку и имеют одинаковые окрестности. Таким образом множества соседей у этих игроков идентичны, т.е. игроки выбирают одинаковые стратегии. Пусть $d_i > d_j$, т.е. игрок i имеет большую окрестность, чем игрок j. Требуется показать, что в симметричном равновесии по Нэшу в сетевой игре на сети, являющейся пороговым графом стратегии игроков i и j будут удовлетворять отношению $x_i > x_j$. Для этого достаточно показать, что $\forall x_i = x_j$, предельные усилия игрока i будут больше усилий игрока j: $h_i > h_j$. Рассмотрим два случая: $g_{ij} = 0$ и $g_{ij} = 1$. В первом случае окрестность игрока j — собственное подмножество окрестности игрока i, т.е. $N_j \subset N_i$. Из-за взаимодополняемости, $h_i > h_j \Rightarrow x_i > x_j$. Во втором случае $N_j \setminus \{i\} \subset N_i \setminus \{j\}$.

Введем обозначения $N_j^- = N_j \setminus \{i\}$ и $N_i^+ = N_i \setminus (N_j \cup \{i\})$. Теперь можно разбить окрестности игроков i и j на непересекающиеся подмножества: $N_j = N_j^- \cup \{i\}$, $N_i = N_j^- \cup N_i^+ \cup \{j\}$. Если оба игрока прикладывают некоторое усилие x, то снова из-за взаимодополняемости,

$h_i > h_j \Rightarrow x_i > x_j$.

Таким образом в симметричном равновесии по Нэшу, стратегии игроков всегда будут удовлетворять условию $x_i > x_j$, если игрок i имеет больше соседей, чем игрок j. □

Результат этой теоремы важен, поскольку он сужает множество равновесий по Нэшу в игре на пороговом графе до симметричным по степени.

2.5. Многошаговая игра

Расширим статическую игру на многошаговую следующим образом. Пусть количество этапов и процесс формирования сети бесконечен, а время дискретно $t = \{1, \ldots, l, \ldots\}$, множество игроков N постоянно. Игра в каждый момент времени t состоит из 3 этапов:

1. Игроки играют в статическую модель, построенную выше.

2. Внешний и потенциально случайный процесс выбирает одного игрока i из сети и решает, должен этот игрок добавить связь или разорвать одну из существующих.

   Более формально:

   внешний процесс выбирает игрока i на этапе t с вероятностью $p_i^t \geq 0$, $\sum_{i=1}^{N} p_i^t = 1$ для любого $t \in \{1, \ldots, l, \ldots\}$;

   вероятность принятия решения о создании связи выбранным игроком i в момент времени t: $q_{i,add}^t \geq 0$, вероятность принятия решения об удалении связи этим же игроком в тот же момент времени: $q_{i,dell}^t \geq 0$. $q_{i,del}^t + q_{i,add}^t = 1$, $\forall t \in \{1, \ldots, l, \ldots\}, \forall i \in \{1, \ldots N\}$.

3. После добавления или удаления связи, игрок может поменять свою стратегию x_i чтобы увеличить выигрыш на данной стадии игры.

В конце каждого периода осуществляются выплаты, и игра повторяется снова.

Рассмотрим внешний случайный процесс. В сетевых играх принято, что создание связи или ее удаление должно происходить с обоюдного согласия игроков. Действительно, во-первых, игрок i, которому предписывается образовать связь, не добавляет эту связь, а предлагает игроку j ее образовать. Но поскольку полезность игроков увеличивается в степени, j-ый игрок никогда не откажет в добавлении связи (i, j). Таким образом ситуация создания связи согласуется с общими принципами формирования сети. Ситуация с разрывом связи противоположная — игрок j, которому i-ый игрок предлагает разорвать связь, никогда на это не согласится. Но в этом случае процесс при ненулевой вероятности образования связи сойдется к полному графе и будет тривиальным. Поэтому внешний процесс «заставляет» игрока i разорвать связь. Здесь происходит расхождение с принципами формирования сетей, но это допущение сделать необходимо. Среди всех игроков, соответственно не имеющих или имеющих связь с выбранным игроком, игрок i сам выбирает игрока j оптимальным образом. Каким образом он это делает, рассмотрим далее.

Во-вторых, если данный процесс предписывает добавить ссылку ведущему игроку, имеющему связь со всеми игроками, то игрок не предпринимает никаких действий, и сразу происходят выплаты, минуя 2 и 3 этапы этого периода. Аналогичная ситуация происходит, если изолированному игроку, не имеющему связь ни с одним другим игроком, пред-

писывается убрать связь, он также не предпринимает действий, и происходят выплаты.

Определение 4. *Функция выигрыша i-ого игрока u_i демонстрирует строгие положительные внешние эффекты, если $\forall i, j$ и $\forall x_j, x'_j$:*
$x_j > x'_j$ выполняется $u_i(x_1, \ldots, x_j, \ldots, x_n, g_i) > u_i(x_1, \ldots, x'_j, \ldots, x_n, g_i)$, в случае, если $g_{ij} = 1$.

Это следует из условия (2).

Введем в рассмотрение граф игры S. S — это граф с множеством вершин N и множеством ребер E. Множество вершин — множество игроков, а E — множество всех пар (i, j), таких, что в матрице смежности g сети (N, g) ячейки с этими координатами ненулевые. Формально: $E = \{(i, j) \mid g_{ij} \neq 0 \ \forall i, j \in N; g_{ij} \in g\}$. Этот граф будет иметь матрицу сопряженности g.

Кроме сети необходимо ввести граф, т.к. в первом разделе данной работы удобнее обращаться к матрице сопряженности, когда происходит анализ функции выигрыша u_i. Когда же рассматриваются последовательности графов, необходимо использовать граф S ввиду строгости изложения.

Взаимно-однозначное соответствие между графом S и матрицей сопряженности g будет сохраняться для всех графов и матриц смежности сети с идентичными индексами.

Теорема 3. *Пусть $S^0, S^1, \ldots, S^l, \ldots$ — последовательность графов многошаговой игры, такая, что граф S^{t+1} образуется из графа S^t $\forall t \in \{0, 1, \ldots, l, \ldots\}$ добавлением или удалением одной связи. Если граф S^0 — пороговый граф, то все графы из последовательности $S^1, \ldots, S^l, \ldots$*

входят в класс пороговых графов, если функция выигрыша $u_i(x; g_i)$ подчиняется условиям (1-4) из раздела (2.2) и демонстрирует строгие положительные внешние эффекты.

Доказательство. Пусть M_i — множество вершин с наибольшей степенью, не соединенных с игроком i, m_i — множество вершин с наименьшей степенью, соединенных с игроком i. Формально:
$M_i = Z_k$, $k = argmax\{d_k \mid g_{ij} = 0, d(j) = d_k\}$;
$m_i = Z_k$, $k = argmin\{d_k \mid g_{ij} = 1, d(j) = d_k\}$.

Граф останется в классе пороговых графов, если игрок i выберет игрока $j \in M_i$, в качестве игрока, с которым он хочет создать связь, или если он выберет игрока $j' \in m_i$, с которым он хочет разорвать связь.

Из определения (3): если $i \in Z_k$, то $M_i = Z_{D+2-k}$, $m_i = Z_{D+1-k}$.

В силу симметричности будем иметь связь: $j \in M_i \Leftrightarrow i \in M_j$, $j \in m_i \Leftrightarrow i \in m_j$.

Введем лемму:

Лемма 3. *Пусть $S^0, S^1, \ldots, S^l, \ldots$ — последовательность графов, такая, что граф S^{t+1} образуется из графа S^t $\forall t \in \{0, 1, \ldots, l, \ldots\}$ добавлением или удалением одной связи. Если граф S^0 — пороговый граф, то все графы из последовательности $S^1, \ldots, S^l, \ldots$ входят в класс пороговых графов, тогда и только тогда, когда выполняются следующие условия:*

1. *Связь (i, j) образуется только, если $j \in M_i$;*

2. *Связь (i, j) разрывается только, если $j \in m_i$.*

Доказательство. Достаточно проверить, что S^1 является пороговым графом тогда и только тогда, когда выполнены условия (1) и (2). Ес-

ли это верно, то по индукции S^{t+1} будет находиться в классе пороговых графов $\forall t \in \mathbb{N}$.

Пусть $Z_1^k, Z_2^k, \ldots, Z_{D^{k+}}^k$ — непересекающиеся разбиение вершин графа S^k, в котором D^{k+} — количество градаций степеней в графе S^k при $k = \{0, 1\}$.

Рассмотрим два узла i в блоке Z_k^0 и j в блоке Z_l^0 в начальном графе S^0.

При добавлении связи (i, j) степени обоих вершин увеличиваются на единицу. Если степень блока Z_{k-1}^0: $d_{k-1} = d_k + 1$, то в графе S^1 вершина $i \in Z_{k-1}^1$. Если степень предыдущего блока в разбиении больше степени блока Z_k^0 больше, чем на единицу, то вершина i образует собственный блок: $\{i\} = Z_k^1$. Аналогично произойдет и с принадлежностью вершины l к блоку: эта вершина либо присоединится к предыдущему блоку в разбиении, либо создаст свой блок, т.е. $j \in Z_{l-1}^1$ или $\{j\} = Z_l^1$.

Остальные блоки в данном распределении не изменяются при переходи от графа S^0 к графу S^1, т.е. $D^{1+} = D^{0+} + c$, где c — количество новых блоков: $c = \{0, 1, 2\}$.

Если выполнено условие (1), то $l = D^{0+} + 2 - k$ и легко проверить, что S^1 является пороговым графом по определению (3) в трех случаях, в зависимости от c. Иначе, если условие (1) не выполняется, т.е. $l \neq D^{0+} + 2 - k$, блоки в S^1 больше не связаны в соответствии с определением.

Аналогично проверяется необходимость и достаточность в случае удаления связи.

□

Теперь осталось показать, что при выполнении условий (1-4) из раздела (2.2), игрок i будет создавать связь с игроком $j \in M_i$, либо удалять

связь с игроком $j' \in m_i$.

По Теореме 2, узлы, находящиеся в блоке M_i — это те узлы, которые не находятся в окрестности N_i, и прикладывают наибольшие усилия среди всех таких узлов. Из-за строгих положительных внешних эффектов, они вносят наибольший вклад в функцию выигрыша игрока i, будучи с ним соединенными.

С другой стороны, узлы, находящиеся в блоке m_i, прикладывают наименьшие усилия среди всех игроков, соединенных с игроком i. И потеря связи уменьшит выигрыш i-ого игрока минимальным образом.

□

Результатом теоремы является факт того, что пороговый граф — поглощающее состояние для сетей такой игры.

Теперь рассмотрим третий этап — изменение стратегии игрока, изменившего структуру сети. Благодаря стратегической взаимодополняемости, связь с игроком $j \in M_i$ увеличит предельную полезность собственных усилий в точке x_i, т.е. игроку i выгодно увеличить свои усилия до x_i', такой, что предельные усилия в этой точке были равны 0. Аналогично с удалением связи: игроку i следует уменьшить усилия, поскольку потеря связи с игроком $j' \in m_i$ уменьшит предельные усилия.

Рассмотрим сходимость сети игры к пороговому графу. Если сеть сойдется к такому графу, то уже не покинет этот граф, как было показано выше.

Теорема 4. *Пусть $S^0, S^1, \ldots, S^l, \ldots$ — последовательность графов многошаговой игры, такая, что граф S^{t+1} образуется из графа S^t $\forall t \in \{0, 1, \ldots, l, \ldots\}$ добавлением или удалением одной связи. Сеть игры сойдется к поглощающему классу пороговых графов, если функция выигры-*

ша $u_i(x; g_i)$ подчиняется условиям (1-4) из раздела (2.2), демонстрирует строгие положительные внешние эффекты и существует $\varepsilon > 0$, такое, что вероятность создания связи p ограничена снизу этой величиной: $p > \varepsilon$.

Доказательство. Рассмотрим некоторый момент времени t. Если в текущий момент времени сеть входит в класс пороговых графов, то она его уже не покинет.

Если же сеть не является пороговым графом, то число связей — некоторое число γ больше 0 и меньше $\frac{n(n-1)}{2}$. Пусть ε — наименьшая вероятность создания связи. Вероятность того, что за следующие γ периодов в сети будут только добавляться связи будет, в худшем случае $\varepsilon^{\frac{n(n-1)}{2}}$. Но это положительное число, и, пусть очень малое, за некоторое конечное количество ходов граф сойдется к полному, и точно попадет в класс пороговых графов.

□

Нестрого говоря, в каждый следующий период динамической игры, сеть подходит все ближе и ближе к классу пороговых графов. Это происходит из-за того, что игроки совершают оптимальный выбор игроков из классов M_i или m_i, чем меняют блоки Z_j вершин. Т.е. в ходе динамической игры не на пороговом графе происходит не «блуждание» по множеству всевозможных графов, а именно сходимость к этому классу. Но, к сожалению, измерить «меру близости» между графом игры и некоторым «похожим» пороговым графом представляется сложной задачей.

Глава 3. Программная реализация модели Лагераса и Зайна

3.1. Формат представления данных

В ходе работы был разработан программный алгоритм на языке программирования Python v3. Этот язык достаточно гибкий, обеспечивает хорошую читаемость кода. Также в открытых репозиториях присутствует большое количество специфических пакетов, значительно расширяющих его возможности. В научной среде он фактически стал одним из основных языков программирования, заменив собой псевдокод, чем убрал границу между программой и алгоритмом.

Для реализации многошаговой игры был реализован класс игры:

```
class game:
    def __init__(self,matrix):
        self.matrix = matrix
        self.players = len(matrix)
        self.p = [1/self.players for i in range(n)]
        self.q = [0.5 for i in range(2)]
```

1. *matrix* — матрица сопряженности сети игры;
2. *players* — количество игроков;
3. p — вектор вероятностей выбора случайным процессом игрока (в данной реализации выбор игроков равновероятен);
4. q — вектор вероятностей выбора случайным процессом действия для выбранного игрока; (в данной реализации вероятности создания или разрыва связи равны на каждом шаге для всех игроков).

3.2. Реализованные функции

Функция *__init__(self,matrix)* — конструктор класса, т.е. функция, создающая объект класса, в данном случае — игру. На вход она получает матрицу смежности. Чтобы не задавать ее громоздкий вид каждый раз при вызове конструктора, были реализованы функции генерации матриц:

1. *createEmpty(n)* — создание матрицы смежности пустого графа;
2. *createFull(n)* — создание матрицы смежности полного графа;
3. *createNSG(n,vector)* — итерационное создание матрицы смежности порогового графа по последовательности *vector*, этот процесс описан в разделе (2.4);
4. *generateRandom(n)* — генерация матрицы смежности случайного графа;
5. *generateRandomNSG(n)* — генерация матрицы смежности случайного порогового графа.

Все матрицы смежности симметричны, не имеют петель (связей, соединяющих вершину саму с собой) и имеют размер n, передаваемый в качестве параметра.

Кроме того, для простоты использования программы, все вышеописанные функции являются обертками над конструктором класса «игра». Это значит, что функция генерация матрицы определенного вида возвращает не матрицу сопряженности сети, а игру с такой матрицей. Например, создание игры с названием *game_Name* со случайной матрицей смежности и количеством вершин 7:

```
game_Name = game.generateRandom(7)
```

Для отображения игры написана функция *show*, для отображения сети игры — функция *show_ graph*. Их исходный код приведен в приложении, т.к. он тривиален и по ходу изложения может быть пропущен.

Так как следует проанализировать сходимость сети к пороговому графу, были написаны 2 функции, проверяющие, является ли граф пороговым или нет:

1. *checkNSG()* — алгоритм проверки сети, основанный на определении порогового графа (3). Строится разбиение вершин Z_j, $j = \{1, \ldots, D^+\}$ и каждая вершина i из Z_k проверяется на смежность с подмножеством $\{Z_1, \ldots, Z_{D^+ + 1 - k}\}$.

2. *checkNSGmy()* — разработанный специально для этой работы алгоритм, основанный на процессе, обратным итерационному процессу формирования сети (обоснование корректности работы этой функции достаточно простое, но алгоритм неэффективный, хотя и находит дополнительную информацию).

Функция *checkNSG()* возвращают 0 или 1, в зависимости от того, является сеть пороговым графом или нет.

Функция *checkNSGmy()* возвращает массив чисел. Первым элементом массива является индикатор графа — число 0 или 1, как и в предыдущей функции (если результат отрицательный, в этом массиве больше нет данных). Если же результат положительный, далее идет последовательность из нулей и единиц, которая соответствует массиву *vector*, который использовался в функции генерации порогового графа. Эта последовательность позволяет проверить корректность алгоритма и построить

граф аналитически с точностью до перестановки вершин при визуализации.

Выше описаны реализации функций генерации сети, проверки сети на отношение к пороговым графам, их визуализации. Осталась функция, имитирующая формирование сети на основе игры, в ходе которой случайный процесс выбирает игрока и действие, которое тот должен совершить. Блок схема реализации этой функции представлена на рис. 2.

В ходе работы программной функции происходит выбор игроков с заданной вероятностью p и q. Проверяется, может ли игрок добавить связь или создать ее, если нет, то выводится сообщение, сигнализирующее об этом. Иначе находятся множества M_i либо m_i и в них выбирается игрок, с которым будет создана или удалена связь.

Связь образуется или разрывается с элементом из M_i или m_i с наименьшим номером, т.к. таких элементов может быть несколько и такой выбор вершины для образования или разрыва связи позволяет точнее предсказывать поведение изменения сети.

После выбора вершины-партнера, изменяется *matrix* данной игры и выводится сообщение с информацией о вершинах-партнерах. В конце каждой итерации выводится граф игры (вообще говоря он может и не измениться).

Вне функции *random_proc()* происходят еще несколько операций: граф проверяется на принадлежность к классу пороговых графов и выводится отчет. Если граф попал в класс пороговых, или прошло заданное количество итераций, а класс так и не был достигнут, процесс завершается.

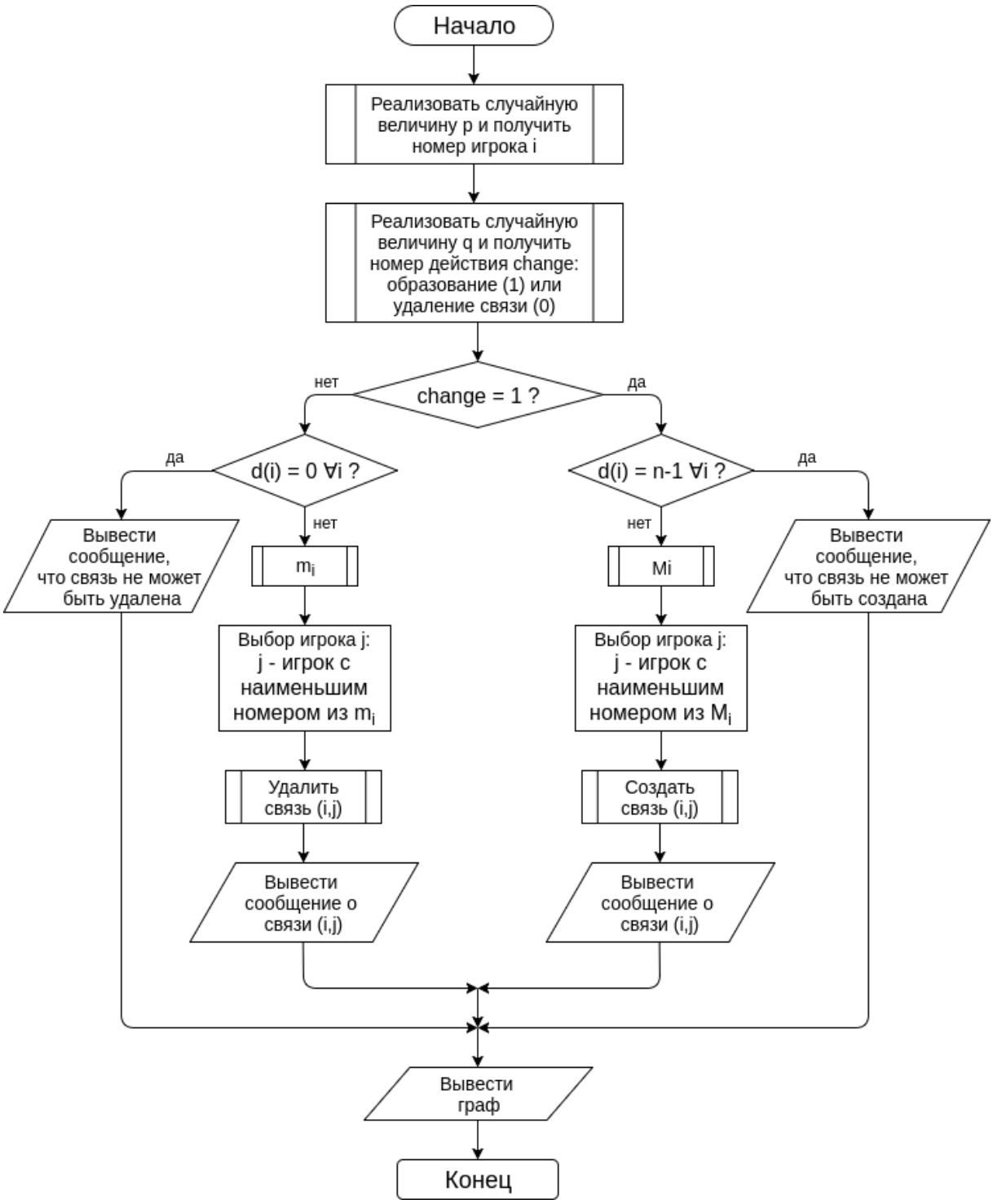


Рис. 2: Блок-схема функции *random_proc()*.

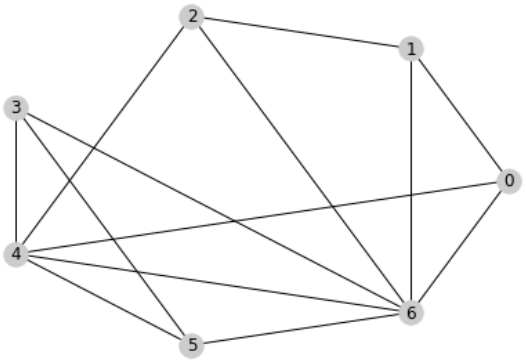
3.3. Примеры

Пример для игры с 7 игроками

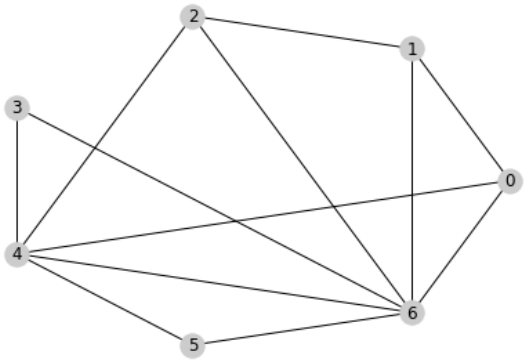
Рассмотрим пример процесса сходимости случайного графа к пороговому, приведенного на рис. 3. Начальная сеть сформирована случайным образом рис. 3а. Вид игры в начальный момент времени:

```
============GAME============
7  players

0 :  [0, 1, 0, 0, 1, 0, 1]
1 :  [1, 0, 1, 0, 0, 0, 1]
2 :  [0, 1, 0, 0, 1, 0, 1]
3 :  [0, 0, 0, 0, 1, 1, 1]
4 :  [1, 0, 1, 1, 0, 1, 1]
5 :  [0, 0, 0, 1, 1, 0, 1]
6 :  [1, 1, 1, 1, 1, 1, 0]

p = [ 0.1429, 0.1429, 0.1429, 0.1429, 0.1429, 0.1429, 0.1429 ]
q = [ 0.5, 0.5 ]
============END=============
```

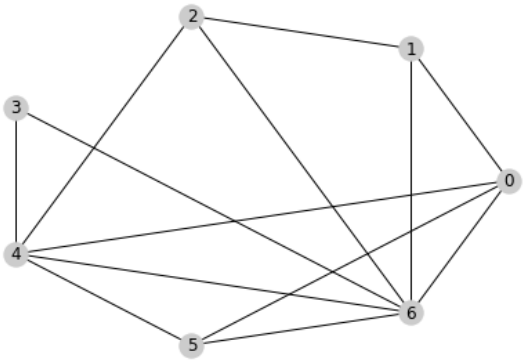
Граф с 7 игроками, не являясь пороговым, сошелся к этому классу за 5 итераций. Графы, находящиеся на рис. 3е-3з, являются пороговыми и, как было установлено в Теореме 3, уже не покинут этот класс.



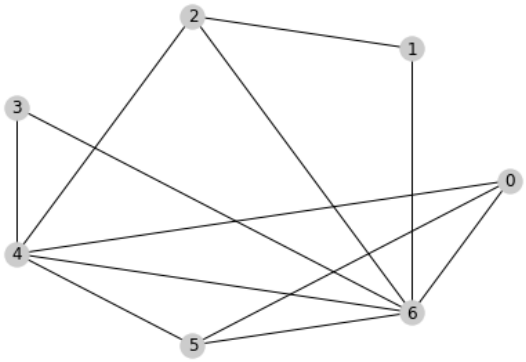
а)



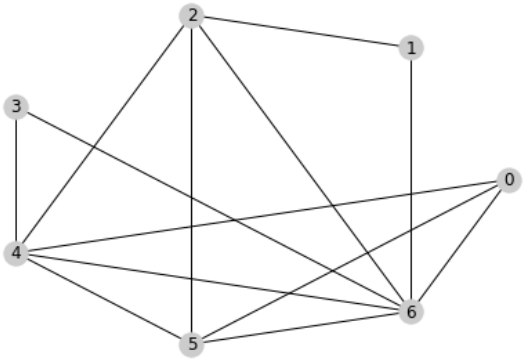
б)



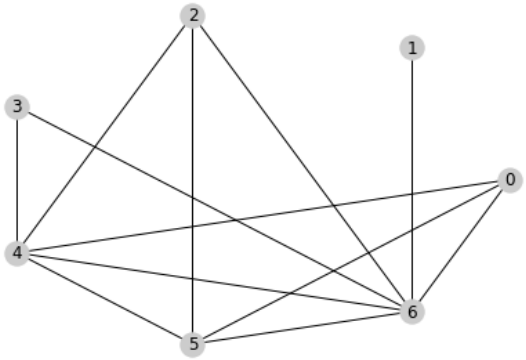
в)



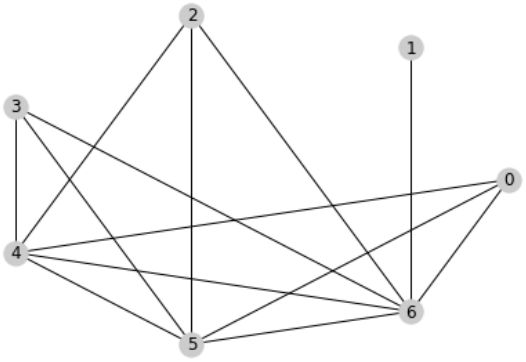
г)



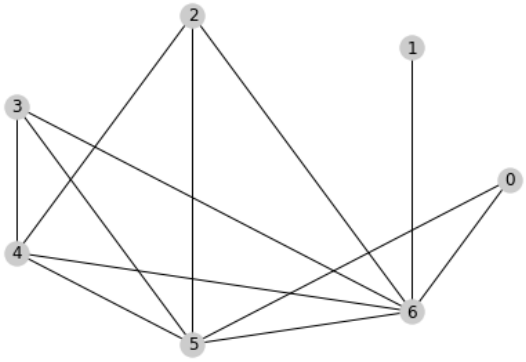
д)



е)



ж)



з)

Рис. 3: Сходимость случайного графа с 7 игроками к пороговому.

Помимо непосредственно графов, в консоль выводится информация, позволяющая судить о динамике изменения сети в ходе итерационного процесса:

```
graph is not NSG
Player  5  remove the link with  3  player
graph is not NSG
Player  5  create the link with  0  player
graph is not NSG
Player  0  remove the link with  1  player
graph is not NSG
Player  5  create the link with  2  player
graph is not NSG
Player  2  remove the link with  1  player
graph is NSG
Player  3  create the link with  5  player
graph is NSG
Player  4  remove the link with  0  player
graph is NSG
```

Как видно, вывод лаконичен и ясен для анализа.

Пример для игры с 15 игроками

В данном случае выкладки становятся слишком большими, поэтому в качестве иллюстрации работы метода будет оставлен начальный граф рис. 4а и три последних рис. 3б-3г, причем последний граф — пороговый. Процесс сошелся. Начальный граф — случайный.

Вид игры в начальный момент времени:

```
============GAME============
15  players

0 :  [0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0]
1 :  [0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0]
2 :  [1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0]
3 :  [0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0]
4 :  [0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0]
5 :  [1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1]
6 :  [1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0]
7 :  [1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1]
8 :  [0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0]
9 :  [0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1]
10 :  [1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0]
11 :  [0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1]
12 :  [0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0]
13 :  [0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0]
14 :  [0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0]

p = [ 0.0667, 0.0667, ... 0.0667, 0.0667 ]
q = [ 0.5, 0.5 ]
============END=============
```

Информация, которая выводится в консоль:

```
graph is not NSG
Player  3  create the link with  5  player
...
graph is not NSG
Player  2  remove the link with  9  player
graph is not NSG
Player  4  create the link with  3  player
graph is not NSG
Player  11  create the link with  9  player
graph is NSG
```

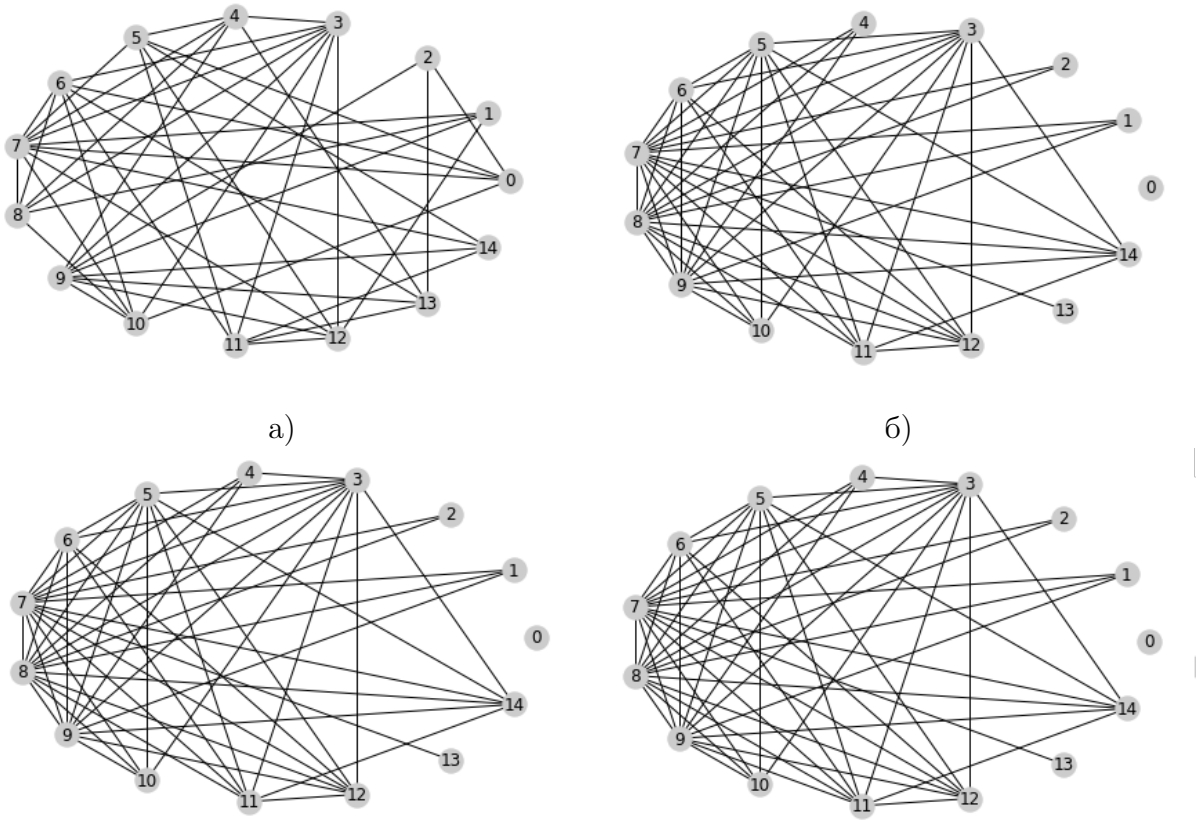


Рис. 4: Сходимость случайного графа к пороговому.

Граф с 15 игроками, не являясь пороговым, сошелся к этому классу за 36 итераций.

Таким образом, подтверждается очевидное свойство сходимости графов к классу пороговых: с увеличением количества вершин, скорость сходимости падает.

Глава 4. Кооперативный вариант игры

4.1. Теоретико-игровая модель кооперативной игры

Рассмотрим кооперативный вариант игры, предложенной Лагерасом и Зайном. В кооперативной модели игроки выбирают такой набор стратегий, который максимизирует их суммарный выигрыш. В такой игре главной задачей является распределение этого выигрыша между игроками. Одним из принципов оптимального распределения суммарного выигрыша является c-ядро, которое может быть пустым. Для построения кооперативного решения необходимо задать характеристическую функцию.

Определение 5. *Пусть $T \subseteq N$ — подмножество игроков. Функцию $v\colon X \times 2^N \to \mathbb{R}$ будем называть характеристической функцией, в смысле Неймана-Моргенштейна, если выполнено условие:*

$$v(x, T) = \max_{i \in T} \min_{j \in N \setminus T} \sum_{i \in T} u_i(x), \tag{9}$$

где X — пространство ситуаций в игре, $x \in X$, $u_i(x)$ — выигрыш игрока i.

Таким образом, функция $v(x, T)$ — максимальный суммарный выигрыш коалиции игроков T, который они могут себе обеспечить при условии, что оставшиеся игроки образуют коалицию $N \setminus T$, которая играет против них.

Определение 6. *Вектор $\alpha = (\alpha_1, \ldots, \alpha_n)$ будем называть дележом, если он удовлетворяет следующим условиям:*

$$\alpha_i \geq v(\{i\}), \forall i \in N \tag{10}$$

$$\sum_{i=1}^{n} \alpha_i = v(N) \tag{11}$$

$\alpha \geq 0, \forall i \in N$; $v(N)$ — *значение характеристической функции для коалиции* N; $v(\{i\})$ — *значение характеристической функции для коалиции из одного игрока* $\{i\} = T$.

Определение 7. *Пусть* α *и* β — *два различных дележа. Дележ* α *будет доминировать дележ* β, *если существует* $T \subseteq N$: $\alpha_i > \beta_i, i \in T$, $\alpha(T) = \sum\limits_{i \in T} \alpha_i \leq v(T)$.

Теперь можно ввести определение *c*-ядра.

Определение 8. *Множество недоминируемых дележей кооперативной игры называется ее c-ядром.*

Определим несколько понятий для построения дальнейшего доказательства.

Определение 9. *Если для* $\forall\ T_1, T_2 \in N$ *и* $T_1 \cap T_2 = \emptyset$ *выполняется условие:*

$$v(x, T_1) + v(x, T_2) \leq v(x, T_1 \cup T_2), \tag{12}$$

то говорят, что характеристическая функция $v(x, T)$ *супераддитивна.*

Введем понятие супермодулярности, при выполнении которого *c*-ядро игры непусто.

Определение 10. *Если для* $\forall\ T_1, T_2 \in N$ *выполняется условие:*

$$v(x, T_1) + v(x, T_2) \leq v(x, T_1 \cup T_2) + v(x, T_1 \cap T_2), \tag{13}$$

то говорят, что характеристическая функция $v(x, T)$ *супермодулярна.*

Определение 11. *Если* $\forall\ T_1 \subset T2 \subseteq N$ *и для* $\forall i \notin T2$ *выполняется следующее условие:*

$$v(x, T_1 \cup i) - v(x, T_1) \leq v(x, T_2 \cup i) - v(x, T_2), \tag{14}$$

то говорят, что игра в характеристической форме проявляет «эффект снежного кома» (snowball effect).

Данное определение введено в [8]. В других источниках ([9], [10]) используется обозначение 0-выпуклой игры. Причем во всех источниках доказывается эквивалентность данного свойства свойству супермодулярности.

4.2. Проверка непустоты c-ядра

В данной работе рассматривается сетевая игра, поэтому необходимо уточнить данную функцию:

$$v(x, T, g) = \max_{i \in T} \min_{j \in N \setminus T} \sum_{i \in T} u_i(x; g_i), \tag{15}$$

таким образом, функция также зависит от сети игры, а именно $v\colon X \times 2^N \times G \to \mathbb{R}$, где G — множество всевозможных сетей.

Игрокам из коалиции $N \setminus T$ для минимизации функции (15) достаточно выбрать нулевые усилия, т.к. по условию (2) предельные усилия игроков не уменьшаются от действий соседа. Это будет эквивалентно тому, что связи (i, j) нет. Следовательно:

$$v(x, T, g) = \max_{i \in T} \sum_{i \in T} u_i(x; g_i^T),$$

где g_i^T — i-ая строка матрицы сопряженности, такая, что в ней $g_{ij}^T = 0$ $\forall i \in T, j \in N \setminus T$ и $g_{ik}^T = g_{ik}$ $\forall i, k \in T$.

Теорема 5. *Пусть функция выигрыша i-ого игрока подчиняется условиям (1-4) из раздела (2.2), а сеть игры является полной. Тогда характеристическая функция $v(x, S, g)$ является супермодулярной.*

Доказательство. Воспользуемся определением (11) и докажем его для нашей теоретико-игровой модели на полной сети.

$v(x, T_1 \cup i, g) - v(x, T_1, g) \leq v(x, T_2 \cup i, g) - v(x, T_2, g)$,

для $\forall\ T_1 \subset T2 \subseteq N$ и для $\forall i \notin T2$.

Напомним, что характеристические функции определены в смысле Неймана-Моргенштейна, что в случае данной игры означает, что все игроки из коалиции, играющей против, выбирают в качестве стратегий нулевые усилия.

Рассмотрим, как увеличатся выигрыши коалиций T_1 и T_2 при входе в коалиции игрока i. $v(x, T_1, g) \leq v(x, T_2, g)$ и $v(x, T_1 \cup i, g) \leq v(x, T_2 \cup i, g)$ следует из супераддитивности, которая выполняется благодаря условию (2). Предельные усилия в соседях в бóльших коалициях на полной сети дают бóльший прирост, чем объединение нескольких разрозненных групп. Сеть полная, значит при входе i в T_1, он будет связан со всеми игроками этой коалиции, и окрестности всех игроков в коалиции будут отличаться разве что отсутствием в них самого игрока, а степени будут равны. С другой стороны, обратную связь получат также все игроки из коалиции. То же самое произойдет и при присоединении i-ого игрока к коалиции T_1. Покажем, что это произойдет в меньших масштабах.

Так как $T_1 \subseteq T_2$, то при добавлении i к этим коалициям, предельные усилия этого игрока в случае T_2 будут больше предельных усилий в случае T_1: $h_i(x, g_i^{T_2}) \geq h_i(x, g_i^{T_2})$. Помимо этого из-за обратных связей $\forall j \in T_1, k \in T_2$: $h_k(x, g_k^{T_2}) \geq h_j(x, g_j^{T_2})$, т.е. каждый игрок из коалиций

получает прирост функции выигрыша, но для игроков из коалиции T_2 он больше. Из отношения предельных усилий непосредственно следует, что $u_k(x, g_k^{T_2 \cup i}) - u_k(x, g_k^{T_2}) \leq u_j(x, g_j^{T_1 \cup i}) - u_j(x, g_j^{T_1}) \; \forall j \in T_1, k \in T_2$.

В совокупности оба этих фактора при условии полной сети показывают 0-выпуклость игры. По доказанным предположениям (4) и (5) из [10] следует, что характеристическая функция игры является выпуклой.

□

Теорема 6. *Пусть функция выигрыша i-ого игрока подчиняется условиям (1-4) из раздела (2.2), а сеть игры является полной. Тогда c-ядро такой игры непусто.*

Доказательство. Из Теореме (5) следует, что характеристическая функция является супермодулярной. Таким образом данная теоретико-игровая модель кооперативной игры также супермодулярной. А по Теореме Шепли [9], c-ядро такой игры непусто. □

В виду общего задания выигрышей $u_i(x; g_i)$, построить такое c-ядро не представляется возможным, но в общем виде показано, что оно не пустое. Также интерес для изучения представляют свойства c-ядер игр на графах, отличных от полного.

Заключение

В результате проделанной работы была проанализирована модель сетевой игры Лагераса и Зайна с одновременным и независимым выбором стратегий и этапом формирования сети под действием внешнего случайного процесса, выбирающего игрока и соответствующее для него действие — добавить или удалить связь.

Была разработана программная реализация процесса формирования сети для любого количества игроков и любой начальной матрицы сопряженности сети. С ростом числа игроков и случайной начальной матрицей сопряженности скорость сходимости графа игры к классу пороговых графов закономерно снижалась. Однако при любом количестве игроков и начальном графе сети, принадлежащему к классу пороговых графов, граф уже не покидал этот класс. Это подтверждает выводы, сделанные в работе Лагераса и Зайна.

В конце работы был проведен анализ c-ядра кооперативной модели данной игры. Было установлено, что для игры на полном графе -ядро не пусто, а характеристическая функция супермодулярна.

В ходе дальнейшей работы планируется построить модели игр с другими методами формирования сети, исключающими внешний случайный процесс или уменьшающими его влияние на игру, тем самым позволяя игрокам взаимодействовать с меньшей долей неопределенности. Также интерес вызывает разработка кооперативных моделей многошаговых игр на основе этой модели, и их возможная визуализация и дальнейший анализ программным алгоритмом.

Список литературы

1. Lagerås A., Seim D. Strategic complementarities, network games and endogenous network formation. // Springer, Game Theory, 2016. p 497-509.

2. Karamardian S. The nonlinear complementarity problem with applications, part 1. // J Optim Theory, 1969a.

3. Karamardian S. The nonlinear complementarity problem with applications, part 2. // J Optim Theory, 1969b.

4. Емеличев В. А, Мельников О. И. Лекции по теории графов. М.: Наука, 1990. 392 с.

5. Уилсон Р. Введение в теорию графов. М.: Мир, 1977. 208 с.

6. Петросян Л. А., Зенкевич Н. А., Шевкопляс Е. В. Теория игр. Спб.:БХВ-Петербург, 2012. 425 с.

7. Petrosyan L., Sedakov A. The Subgame-Consistent Shapley Value for Dynamic Network Games with Shock // Springer, Games Appl., 2016. p 520-537.

8. Демешев Б.Б. Лекции по кооперативной теории игр. https://github.com/bdemeshev/gt201/wiki

9. Розенмюллер И. Кооперативные игры и рынки. М.: Мир, 1973. 160 с.

10. Мулен Э. Кооперативное принятие решений: аксиомы и модели. М.: Мир, 1991. 464 с.

Приложение

Приложение 1. Программная реализация модели Лагераса и Зайна

```
 1 # -*- coding: utf-8 -*-
 2
 3 import random as rn
 4 import networkx as nx
 5 import matplotlib.pyplot as plt
 6 import copy
 7
 8 class game:
 9     def __init__(self,matrix):
10         self.matrix = matrix
11         self.players = len(matrix)
12         self.p = 1/self.players
13         self.q = 0.5
14     def show(self):
15         print('============GAME============')
16         print(self.players,' players')
17         print()
18         for i in range(self.players):
19             print(i,': ',self.matrix[i])
20         print()
21         print('p = ',self.p)
22         print('q = ',self.q)
23         print('============END=============')
24
25     def createEmpty(n):
26         matrix = []
27         for i in range(n):
28             matrix.append( [] )
29             for j in range(n):
30                 matrix[i].append( 0 )
31         return game(matrix)
```

```
    def createFull(n):
        matrix = []
        for i in range(n):
            matrix.append( [] )
            for j in range(n):
                if i==j:
                    matrix[i].append( 0 )
                else:
                    matrix[i].append( 1 )
        return game(matrix)
    def createNSG(n,vector):
        if len(vector) < n:
            for i in range(len(vector),n):
                vector.append(0)
        tempGame = game.createEmpty(n)
        for i in range(0,tempGame.players):
            if vector[i] == 0:
                pass
            else:
                for j in range(i):
                    tempGame.matrix[i][j] = tempGame
.matrix[j][i] = 1
        return tempGame

    def generateRandom(n):
        tempGame = game.createEmpty(n)
        for i in range(tempGame.players):
            for j in range(i):
                tempGame.matrix[i][j] = rn.randint(0
,1)
                tempGame.matrix[j][i] = rn.randint(0
,1)
        return tempGame
    def generateRandomNSG(n):
```

```
63          probability = 0.5
64          vectorVertice = []
65          for i in range(n):
66              prob = rn.uniform(0,1)
67              if prob >= probability:
68                  vectorVertice.append(0)
69              else:
70                  vectorVertice.append(1)
71          return game.createNSG(n,vectorVertice)
72
73      def checkNSG(self): #return 0 if NSG
74          n = self.players
75          matrix = self.matrix
76          all_degrees = []
77          unique_degrees = []
78          D = {}
79          for i in range(n):
80              all_degrees.append( sum(matrix[i]) )
81              if all_degrees[i] == 0:
82                  continue
83              try:
84                  D[all_degrees[i]].append( i )
85              except KeyError:
86                  D[all_degrees[i]] = []
87                  D[all_degrees[i]].append( i )

88          unique_degrees = list(set(all_degrees))
89          unique_degrees.sort()
90          if unique_degrees[0] == 0:
91              unique_degrees = unique_degrees[1:]
92          unique_degrees.reverse()
93          D_plus = len(unique_degrees)
94          if unique_degrees == []:
95              return 0
```

```
    96         if (len(unique_degrees) == 1)and(unique_degr
ees[0] == n-1):
    97             return 0
    98         for i in range(len(unique_degrees)):
    99             u = []
   100             for j in range(0,D_plus-i):
   101                 u.extend( D[unique_degrees[j]] )
   102             v = D[unique_degrees[i]]
   103             for k in v:
   104                 count = 0
   105                 for j in u:
   106                     if j == k:
   107                         continue
   108                     else:
   109                         count += 1
   110                     if matrix[j][k] != matrix[k][j]:
   111                         return 1
   112                     if matrix[j][k] == 0:
   113                         return 1
   114                 if count != all_degrees[k]:
   115                     return 1
   116         return 0
   117     def checkNSGmy(self):
   118         n = self.players
   119         matrix = copy.deepcopy(self.matrix)
   120         vector = []
   121         full_vector = []
   122         full_vector_label = []
   123         max_degrees = 0
   124         count = 0
   125         while len(matrix)>0:
   126             if len(matrix) == count:
   127                 return [1]
   128             count = n = len(matrix)
```

```
129             for i in range(n-1,-1,-1):
130                 if sum(matrix[i]) == 0:
131                     vector.append(i)
132                     matrix.pop(i)
133             for i in range(len(matrix)):
134                 for k in vector:
135                     matrix[i].pop(k)
136             full_vector.extend(vector)
137             for i in range(len(vector)):
138                 full_vector_label.append('isolated')
139             vector = []
140 
141             n = len(matrix)
142             max_degrees = n-1
143             for i in range(n-1,-1,-1):
144                 if sum(matrix[i]) == max_degrees:
145                     vector.append(i)
146                     matrix.pop(i)
147             for i in range(len(matrix)):
148                 for k in vector:
149                     matrix[i].pop(k)
150             full_vector.extend(vector)
151             for i in range(len(vector)):
152                 full_vector_label.append('friendly')
153             vector = []
154         full_vector_label.append(0)
155         full_vector_label.reverse()
156         return full_vector_label
157 
158     def random_proc(self):
159         n = self.players
160         matrix = self.matrix
161         p = rn.randint(0,self.players-1)
162         q = rn.randint(0,1)
```

```
163         if q == 0:
164             m = []
165             degrees = []
166             gamers = []
167             for i in range(n):
168                 if i == p: continue;
169                 if matrix[i][p] == 1:
170                     m.append( i )
171                     degrees.append( sum(matrix[i]) )
172             if len(m) == 0:
173                 print('Player ',p,' must remove the
link, but he cant')
174                 return 0
175             min_d = min(degrees)
176             for i in range(len(degrees)):
177                 if degrees[i] == min_d:
178                     gamers.append(m[i])
179             gamer = gamers[0]
180             #gamer = degrees.index(min_d)
181             self.matrix[p][gamer] = self.matrix[game
r][p] = 0
182             print('Player ',p,' remove the link with
 ',gamer,' player')
183         else:
184             M = []
185             degrees = []
186             gamers = []
187             for i in range(n):
188                 if i == p: continue;
189                 if matrix[i][p] == 0:
190                     M.append( i )
191                     degrees.append( sum(matrix[i]) )
192             if len(M) == 0:
193                 print('Player ',p,' must create the
```

```
link, but he cant')
   194                 return 0
   195             max_d = max(degrees)
   196             for i in range(len(degrees)):
   197                 if degrees[i] == max_d:
   198                     gamers.append(M[i])
   199             gamer = gamers[0]
   200             #gamer = degrees.index(max_d)
   201             self.matrix[p][gamer] = self.matrix[game
r][p] = 1
   202             print('Player ',p,' create the link with
 ',gamer,' player')
   203
   204     def show_graph(self):
   205         temp_g = nx.Graph()
   206         temp_list = []
   207         for i in range(self.players):
   208             temp_g.add_node(i)
   209             for j in range(self.players):
   210                 if self.matrix[i][j] != 0:
   211                     temp_list.append( (i,j,self.matr
ix[i][j]) )
   212         temp_g.add_weighted_edges_from(temp_list)

   213         plt.figure(3,figsize=(6,4))
   214         grey_color =
   215         nx.draw_circular(temp_g,with_labels=True,nod
e_color=grey_color)
   216         plt.show()
   217         return temp_g
   218
   219 n = 7
   220 step = 10
   221 game1 = game.generateRandom(n)
```

```
222 times = 0;
223 for i in range(step):
224     game1.show_graph()
225     result = game1.checkNSG()
226     if result == 0:
227         print('graph is NSG')
228     else:
229         print('graph is not NSG')
230         times += 1
231     game1.random_proc()
232 print(times,'from',i)
```